NUMERIK / MAYER URSULA

1. Einführung Matlab

Grundlegende Bedienung von MATLAB

KEYBOARD SHORTCUTS

Ctrl + C	Interrupt code
F5	Run code

MATLAB-Befehle und -Funktionen

lookfor	Stichwort suchen
help	Hilfefunktion
doc	Öffnet Dokumentation
primes	Generate list of prime numbers
clear all	Löscht Workspace
clear A	Löscht A aus Workspace
clc	Cleans command window
who	Lists the variables in the current workspace
whos	List current variables, long form
ans	Most recent answer

HELP, DOCUMENTATION, SHORT CUTS

>> lookfor & doc	Further explaning
>> help ops	operations
>> help elfun	Elementary, math functions
>> help specfun	Special functions
>> help graphics	Basic graphic functions
>> help graph2d/ graph3d	Graphic functions for 2- / 3D
>> help elmat	Elementary Matrix-/ functions
>> type <filename>	Display content of file
>> numel (A)	Number of elements
↑ , ↓ (statatur)	Using previous commands
>> clc	Delete command window
>> home	Lift command window
>> clear / clear all	Delete workspace
>> clear <name>	Delete variable in workspace
>> close /clf	Close figure window
>> close all	Close all figure windows
>> who	List all variables
>> whos	List all variables with specifications
>> open <filename>	Open / create if not existent
>> edit <filename>.m	
>> diary <filename>	Start saving commands
>> diary	Finish saving & create file
>> echodemo <filename>	Open & run script step by step
"RUN SECTION"	same

DATA IMPORT/EXPORT

xlsread/xlswrite	Read/write Excel
dlm-read/write	
load/save – ascii	Save as text file (.txt, csv) *save('filename.txt','var1', 'var2','-ascii')*
webread [URL]	Read content from URL
websave [URL]	Save conent from URL
web [URL]	Open [URL]
audioread	Open MP3/4 file [Y,FS]

Grundrechenarten & Darstellungsformat

Addition	Subtraktion	Multiplikation	Division
+	-	*	/

SEQUENCE OF OPERATOR

1. () {} []
2. .' .^ ' ^
3. + - ~
4. .* ./ .\

DISPLAY FUNCTION

;	Don't show command in command window
,	separate commands in line
…	Continue command in next line
1e3	1000
format long	3.141592653589793
Format longE/Eng	3.141592653589793e+00
format short	3.1416
format shortE/Eng	3.1416+00
format rat/rational	355/113
format shortG	Use what best fits
bank	3.14 (2 digits)
format compact	**Suppresses extra line**
format loose	Back to usual distance

BASIC MATH FUNCTIONS

abs(x)	Absolute value
sqrt (x)	**Square**
sum(x)	Sum
sumsum (x)	Cumulative sum
factorial (x)	!
factor(A)	Prime factors (faktorisieren)
round (x,n)	Rounding
ceil (x)	Aufrunden
fix (x)	Abrunden
log (x)	Natural logarithm - *ln (x)*
log10(x)	Common logarithm
rem (a,b)	Rest of division (if existing)
mod (a,b)	"" Modulo operation
lcm(a,b)	Least common multiples
gcd (a,b)	Greatest common multiples
nthroot (a,n)	Real n-th root of a
min (A)	Minimum value
cummin(A)	Cumulative minimum
max (A)	Max value
cummax(A)	Cumulative maximum
complex (a,b)	Complex Array *(z= a+bi)*
real (x)	Real part of complex n
image (x)	Imaginary part of complex n
angle (x)	Phase-Angle of complex in rad

Elementare Funktionen

Exponential- und Logarithmusfunktion

exp	der Eulerschen Zahl e
log	natürliche Logarithmus

exp (log(x)) = x

log10	dekadische Logarithmus (zur Basis 10)

Formel für beliebige Basen:

$$\log_b(x) = \frac{\log_b(x)\log_a(b)}{\log_a(b)} = \frac{\log_a\left(b^{\log_b(x)}\right)}{\log_a(b)} = \frac{\log_a(x)}{\log_a(b)}$$

Potenzen und Wurzeln

Nichtnegative Basis a ≥ 0 & reeller Exponent b ∈ R: a^b := **exp(b log a)** (für a < 0 ist die Situation komplizierter)

^ oder **power**	Potenzen

Für eine natürliche Zahl n ∈ N mit n ≥ 1 und eine nichtnegative reele Zahl, a ∈ R mit a ≥ 0, ist die n-te Wurzel aus a, $\sqrt[n]{a}$, definiert als die Lösung der Gleichung $x^n = a$.

Nthroot oder **sqrt** (n=2)	Wurzeln

Trigonometrische und weitere Funktionen

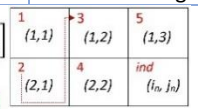
sin	Sine of argument in radians
sind	Sine of argument in degrees
cos	Cosine of argument in radians
cosd	Cosine of argument in degrees
tan	Tangent of argument in radians
tand	Tangent of argument in degrees
abs	Absolute value
sign	Signum function. *For each element of X, sign(X) returns 1 if the element is greater than zero, 0 if it equals zero and -1 if it is less than zero. For the nonzero elements of complex X, sign(X) = X ./ ABS(X).*
ceil	Round towards plus infinity
fix	Round towards zero
floor	Round towards minus infinity
round	Rounds towards nearest decimal or integer

TRIGONOMETRIC FUNCTIONS

# → sin,cos,tan,sec,cot	… secant, cotangent
#/#d(x)	# In **radians / degrees**
#h(x)	Hyperbolic of #
a#/a#d(x)	Inverse of # in **rad/ deg [arc#]**
a#h(x)	Inverse Hyperbolic of #
atan2/atan2d (x)	Four-quadrant inverse tan “”
hypot (x)	Square root of sum of square
deg2rad (x)	From degrees to radians
rad2deg (x)	Radians to degrees

$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} \times \begin{bmatrix}7\\ 8\\ 9\end{bmatrix} = \begin{bmatrix}50\\ 122\end{bmatrix}$$

$2 \times 3 \quad 3 \times 1 = 2 \times 1$



Vektoren und Matrizen

Vektoren

Vektor v ∈ R ^{n}	**transponierte Vektor v ^{T}**
Spaltenform (vertikal)	Zeilenform (horizontal)
$v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$	$v^\top = \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix}$

>> v = [2;3;5;8]	**>> v = [2 3 5 8]**

Vektoren und andere Zahlenschemas (arrays) mit eckigen Klammern begrenzt

v(i)	Die i-te Komponente eines Vektors v

*reelle Zahl vi ∈ R, i = 1, . . . , n

length(v)	Die Anzahl der Komponenten (n) eines Vektors v
‘ / transpose	Vektor und Matrizen Transponieren

Matrizen

Matrix **A** ∈ R $^{m \times n}$ mit *m* Zeilen und *n* Spalten

>> A = [2 3 4;5 6 7;0 1 9;2 4 7]

Dimension einer Matrix A ∈ R $^{m \times n}$	Den **ij-ten Eintrag** der Matrix A
die Zahlen m und n	die Zahl a_{ij} ∈ R, i = 1, . . . , m, j = 1, . . . , n
size(A)	**A(i,j)**

Transponierte A> ∈ R $^{n \times m}$ der Matrix A ∈ R $^{m \times n}$

eye	Identity matrix (Einheitsmatrix)
zeros(N)	is an N-by-N matrix of zeros
ones(N)	is an N-by-N matrix of ones
rand	Uniformly distributed pseudorandom numbers
rand(N)	returns an N-by-N matrix containing pseudorandom values
diag	Diagonal matrices and diagonals of a matrix
fliplr	Flip array in left/right direction
flipud	Flip array in up/down direction
spy	Visualize sparsity pattern
spy(S)	plots the sparsity pattern of the matrix S

Operationen:

Addition (v + w oder A + B) und **Multiplikation mit einem Skalar** (v * w oder A * B) von Vektoren und Matrizen Komponentenweise.

Multiplikation von Matrizen und Vektoren

Bedingung → $\underset{m \times n}{\underline{A}} \quad \underset{n \times p}{\underline{B}} = \underset{m \times p}{\underline{C}}$

$$c_{ij} := a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}, \quad i = 1, \ldots, m, \; j = 1, \ldots, p$$

Zusammenhang mit dem Skalarprodukt zweier Vektoren →

$$\underset{1 \times 1}{v \cdot w} := \sum_{k=1}^{n} v_i w_j \overset{(2)}{=} \underset{1 \times n}{v^\top} \underset{n \times 1}{w}$$

Produkt zweier Vektoren mit n Komponenten → (n × n)-Matrix

$$\underset{n \times 1}{v} \quad \underset{1 \times n}{w^\top} \overset{(2)}{=} \underset{n \times n}{\begin{pmatrix} v_1 w_1 & \cdots & v_1 w_n \\ \vdots & v_i w_j & \vdots \\ v_n w_1 & \cdots & v_n w_n \end{pmatrix}}$$

Produkt einer (m × n)-Matrix mit einem Vektor mit n Komponenten →

$$\underset{m \times n}{\underline{A}} \quad \underset{n \times 1}{v} \overset{(2)}{=} \underset{m \times 1}{\begin{pmatrix} \sum_{k=1}^{n} a_{1k} v_k \\ \vdots \\ \sum_{k=1}^{n} a_{ik} v_k \\ \vdots \\ \sum_{k=1}^{n} a_{mk} v_k \end{pmatrix}} = \sum_{k=1}^{n} \begin{pmatrix} a_{1k} \\ \vdots \\ a_{ik} \\ \vdots \\ a_{mk} \end{pmatrix} v_k$$

$$= v_1 \begin{pmatrix} a_{11} \\ \vdots \\ a_{m1} \end{pmatrix} + v_2 \begin{pmatrix} a_{12} \\ \vdots \\ a_{m2} \end{pmatrix} + \cdots + v_n \begin{pmatrix} a_{1n} \\ \vdots \\ a_{mn} \end{pmatrix}$$

$$\underline{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \quad \underline{A}^\top = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{pmatrix}$$

* Summation über den Index k von 1 bis n

Vektoroperationen

Addition von Vektoren sowie die Multiplikation eines Vektors mit einem Skalar:

3*v	-w	(-1)*w
v+w	v-w	v+(-w)

verschiedene Produkte:

v*w	4x1-Matrix multipliziert mit 4x1-Matrix → **Error**
v.'*w	1x4-Matrix multipliziert mit 4x1-Matrix (ganze Zahl)
dot(v,w)	Ganze Zahl = v.'*w
v*w.'	4x1-Matrix multipliziert mit 1x4-Matrix

Die meisten Funktionen operieren komponentenweise auf Vektoren → Falls die Funktionsweise nicht eindeutig ist, muss der Punkt (.) verwendet werden.

: oder colon	Vektoren mit konstantem Abstand
1:10	1 2 3 4 5 6 7 8 9 10
1:2:10	1 3 5 7 9
reshape	Reshape array
reshape(X,M,N)	returns the M-by-N matrix whose elements are taken column wise from X. *An error results if X does not have M*N elements.*

Matrixoperationen

MATRIX OPERATIONS	
svd(A)	Singular values → diagonalisieren
norm(A)	Norms (magnitude of vector)
det(A)	Determinant of square matrix
factor(A)	Prime factors
eig(A)	Eigenvalues and eigenvectors
inv (A)	Inverse
diag(A)	Elements in the principal diagonal
tril(A)	Lower triangular part of A
triu(A)	Upper triangular part of A
prod(A)	Product of array elements
cumprod(A)	Cumulative product
sort(A)	Sorts from smallest to largest
sortrows(A)	Sorts rows of A in ascending order
rank(A)	Rank
chol(A)	Cholesky factorization of matrix

Matrix und Vektoren

VEKTORS & MATRICES	
A= [1 2 3]	1 × 3 vector (double array).
A= [1; 2; 3]	3 × 1 vector
A= [1 2; 3 4]	2 × 2 matrix (double array)
A= [1 2 ;3 4 ; 5 6]	3 × 2 matrix
A(:, c_i)	Elements of colum c_i
A(:, [c_n, c_m,…])	Elements of colums c_n, c_m, …
A(r_n,:)	Elements of row r_i
A([r_n,r_m,…], :)	Elements of rows r_n, r_m, …
A(:)	All elements → *displayed separately (create a Vektor with all Elements)*
A(i, j)	Element *(i, j)* of A
A(r_n:r_m, c_n:c_m) **A([r_n,r_m,…], [c_n,c_m,…])**	Elements in rows from r_n to r_m which are in columns from c_n to c_m
A(ind)= e	Replace element *i-th (index)* with e
A(i, j) = e	Replace element *(i, j)* with e
A(:, c_i)=[c_{new}]	Replace column c_i with [a;b] →must be same size → ***add column if not existent***
A(r_i, :)=[ ]	Delete row r_i → *vice versa for c_i*
A([r_n : r_m], :) = [ ] **A([r_n, r_m,…], :) = [ ]**	Delete rows r_n, r_m, … → *vice versa for c_n, c_m*
ones(a,b)	a × b matrix of 1 values
zeros(a,b)	a × b matrix of 0 values
eye(a)	Identity matrix of size a
Size () / Length()	Matrix dimension/länge
x= {1, 'st' }	1 × 2 cell array
sx.x1 = [1 2 3]	1 × 3 vect. stored in sx structure array
sx.x2 = {1, 'st'}	1 × 2 cell stored in sx structure array
A(ind)	*Element i-th of A*
A(i_n : i_m) ***A([i_n, i_m,..])***	*Elements from i_n to i_m → index*
A(i:end)	*Elements from the i-th to the last one*
A(i:k:end)	*Every k-th element to the last one*

Linear Reression, Function, Solver

LINEAR REGRESSION	
Fitlm(x,y)	Fit linear regression
fitglm(x,y)	Fit generalized linear regression
stepwiselm(x,y)	Fit stepwise linear regression

FUNCTION → FILENAME == FUNCTIONNAME

```
function [t, sinf, cosf, expf] = functionname(f1, f2,f3)
t = (0:0.1:2);
sinf = sin(2*pi*f1*t);
cosf = cos(2*pi*f2*t);
expf =exp(f3*t);

plot(t, [sinf; cosf; expf])
xlabel('text'), ylabel('text')
title ('text')
end
```

```
>> functionname (15, 6, 3)
```

Function Handles

```
f = @(x) sin(x.^2)./(5*x);

f(pi/2)
    0.0795
f([-pi/2, 0, pi/2])
   -0.0795 NaN 0.0795
```

SOLVER

```
syms a b c x
eqn = x*a^5 == 31*(b/c);
S = solve(eqn,x)
```

Loops and Conditional Statements

Conditional Statements

```
if a > 10
    disp('Greater than 10');
elseif a == 5
    disp('a is 5');
else
    disp('Neither condition met');
end
```

For loops

```
for k = 1:5
    disp(k);
end
```

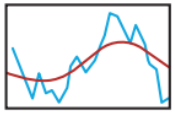
While loops

```
k = 0;
while k < 7
    k = k + 1;
end
```

Plots

PLOTS	
Examples: *echodemo graf2d & echodemo graf2d2*	
figure	Open a new figure
plot(x , y, 'ColorSimbol' **plot(t , [f_1 ;f_2 ;f_3])**	
axis normal /tight/equal/ square/ fill **axis ([x_a,x_b,y_a,y_b])**	
title('*text*')	
x-/y-/zlabel('*text*')	
legend('*v*','*w*') / legend toggle	
grid on/off	Gitter
box on/off	
x-/y-/zlim([min,max])	
hold on/off	
text(x,y, '*text*')	Add text to a point
datetick('x',fm)	Date formatted tick labels (fm is format)
xtickformat(fm)	X-axis label format
ytickformat(fm)	Y-axis label format.

Smooth Data >>

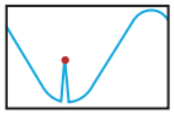


```
B = smoothdata(A,method);
```

Smooth noisy data with methods:

```
'movmean','movmedian','gaussian',
'lowess','loess','rlowess',
'rloess','sgolay'
```

Detect Outliers >>

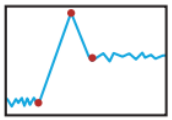


```
TF = isoutlier(A,method);
```

Identify outliers with methods:

```
'median','mean','quartiles',
'grubbs','gesd'
```

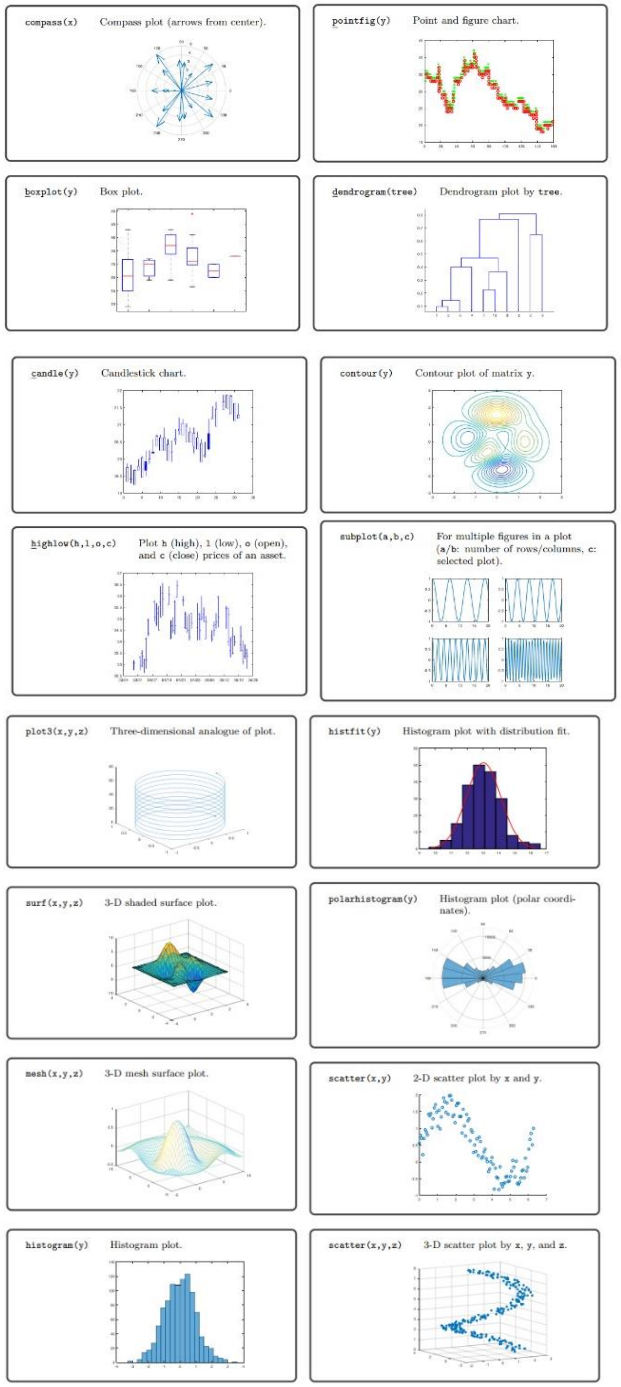
Detect Change Points >>

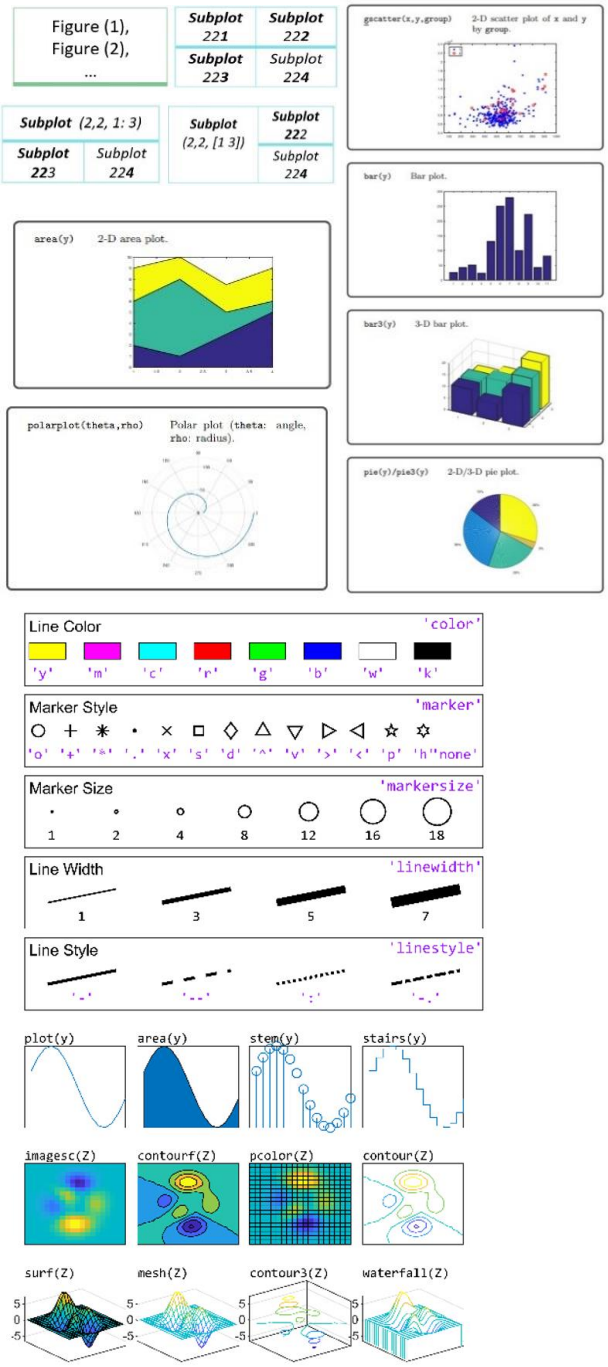


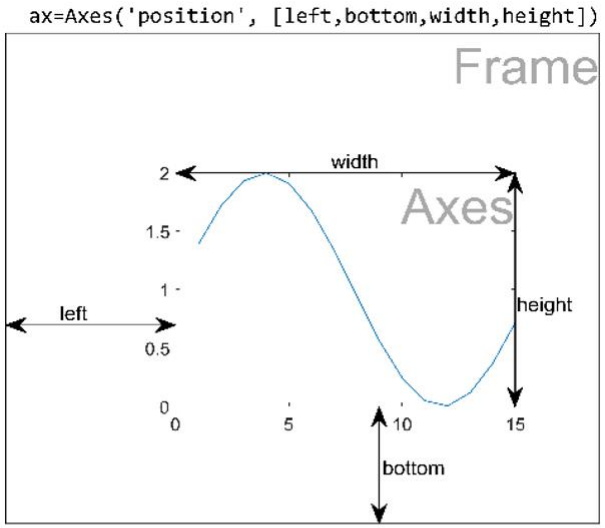
```
TF = ischange(A,method);
```

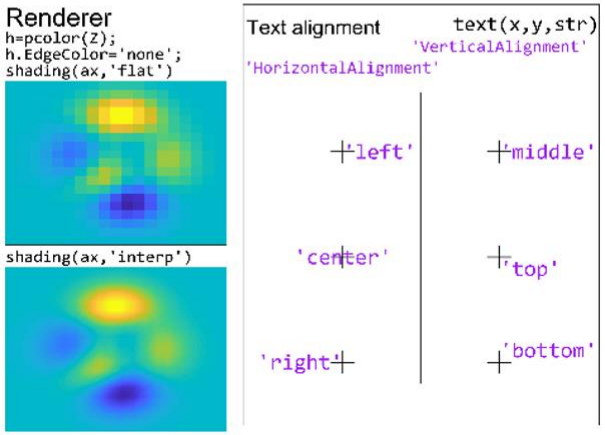
Find abrupt changes with methods:

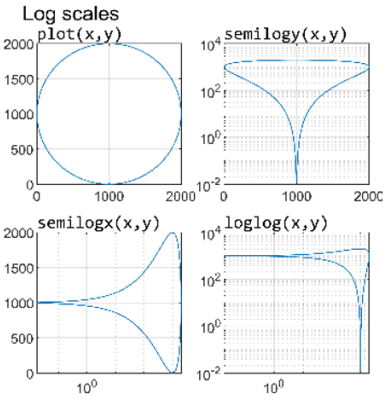
```
'mean','variance','linear'
```

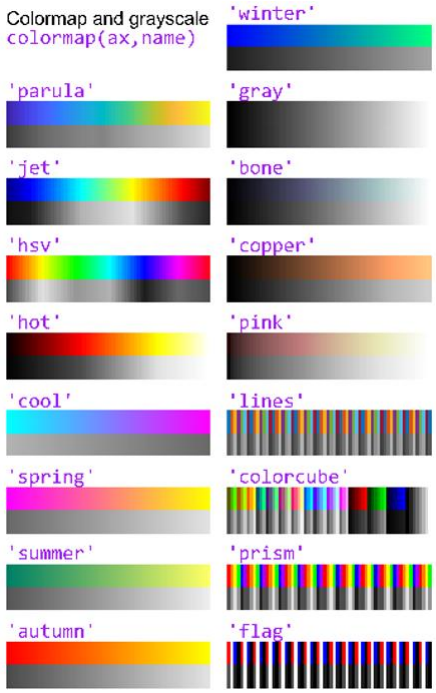








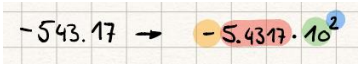




2. Endliche Arithmetik

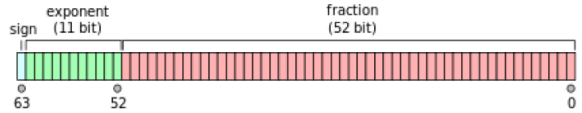
Darstellung von Zahlen

Normalisierte Zahl in Matlab → Produkt mit 10er Potenz

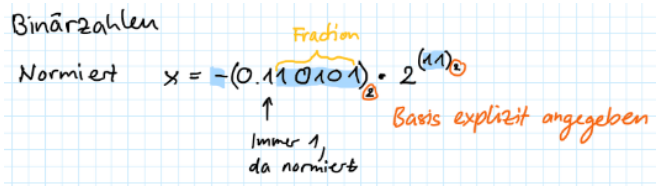


Computer legt Zahl auf 64-Bit Speichereinheit ab.

1 Bit: Vorzeichen, 11 Bit: Exponent, 52 Bit: Mantisse / Fraction



Binär und Dezimal Darstellung	
bin2dec('x')	Binär in Dezimalsystem
dec2bin(x)	Dezimal in Binärsystem
eps	Epsilon



Gleitkommazahlen doppelter Genauigkeit und Format

Format	
Matlab default	64-bit Gleitkommazahlen
Darstellung	`help format`
Default Format	`format short`
Andere Formate	`format long, short e, long e, rat`

Konsequenzen endlicher Arithmetik

- Darstellungsfehler in Daten z.B.: `pi`
- Binäre Darstellung von 0.1 hat einen Fehler
- Rundungsfehler: $\left(\frac{1}{3}\right)^3 - \frac{1}{27} \neq 0!$
- Overflow: 2^{10124} → `Inf`
- Addition und Subtraktion `1+1e-20`
- Mantissen werden addiert, *nachdem* der kleinere Exponent angeglichen wurde auf den grösseren (durch Kommaverschiebung)
- Die Rundung in der Darstellung der Mantisse erfolgt unabhängig vom Exponenten. Daraus folgt, dass der Abstand von Fliesskommazahlen mit dem Exponenten zunimmt.
- Abstand zweier benachbarter Gleitkommazahlen (Maschinenepsilon) eps ist die kleinste Zahl welche: $1 + eps > 1$
  `eps(1)` *entspricht* 2^{-52}
- Abbruchfehler `sin(pi) ~= 0`

Logik Syntax

Logik Syntax	
Wahr	1

Falsch	`0`
Gleich	`==`
Ungleich (nicht)	`~=`
Grösser/kleiner gleich	`>/<  >=/<=`
Logisches oder	`zahl<10 | zahl>20`

if Bedingung

```
if ...
    ...
else
    ...
end
```

for Schleife *Anzahl von Durchführungen*

```
for k = 1:20
    disp('Hallo')
end
```

while Schleife

```
while ...
    ...
End
```

Andere Befehle	
Abfrage	`input()`
Division mit Rest	`mod(a,2)`
Zahl in String	`num2string()`
Abrunden	`floor()`
Betrag	`abs()`
abs()	`%%`

3. Polynome, Potenzreihen und Taylor-Approximation

Taylor Polynome

Taylor-Reihe einer oft genug differenzierbaren Funktion f am Entwicklungspunkt x_0:

$$t(x) = f(x_0) + \frac{f'(x_0)}{1!}\cdot(x-x_0) + \frac{f''(x_0)}{2!}\cdot(x-x_0)^2 + \frac{f'''(x_0)}{3!}\cdot(x-x_0)^3 + \cdots$$
$$= a_0 + a_1\cdot(x-x_0) + a_2\cdot(x-x_0)^2 + a_3\cdot(x-x_0)^3 + \cdots$$

k-tes Taylor-Polynom

einer oft genug differenzierbaren Funktion f am Entwicklungspunkt x_0:

$$t_k(x) = f(x_0) + \frac{f'(x_0)}{1!}\cdot(x-x_0) + \frac{f''(x_0)}{2!}\cdot(x-x_0)^2 + \cdots + \frac{f^{(k)}(x_0)}{k!}\cdot(x-x_0)^k$$

Fehlerschätzung per Restglied

Formale Restglied-Darstellung nach Lagrange

mit Zwischenstelle z für exakten Fehler:

$$f(x) - t_k(x) = \frac{f^{(k+1)}(z)}{(k+1)!}(x-x_0)^{k+1}$$

Dabei ist z aus dem Bereich zwischen x und x_0 aber unbekannt. Auch ohne Kenntnis des genauen Werts z kann man aber den Fehler abschätzen, falls man die entsprechende Ableitung begrenzen kann.

Fehlerschranke

falls $|f^{(k+1)}| \leq M$ im Intervall zwischen x und x_0, dann

$$|f(x) - t_k(x)| \leq \max_z \left|\frac{f^{(k+1)}(z)}{(k+1)!}(x-x_0)^{k+1}\right| = \left|\frac{M}{(k+1)!}(x-x_0)^{k+1}\right|$$

Interpretation: Abweichung des Taylorpolynoms von der Funktion wird begrenzt einerseits durch Maximum M der nächsthöheren Ableitung und andererseits die Entfernung vom Entwicklungspunkt.

`n!`	`factorial(n)`

Linearisierung an einem Entwicklungspunkt

Ersetze kompliziertes f durch lineare Funktion in der Nähe von x_0. Entspricht der Verwendung von:

$$f(x) \approx f(x_0) + f'(x_0)\cdot(x-x_0)$$

Vielfache Anwendungen dieser Idee: Stabilität von Gleichgewichtspunkten dynamischer Systeme, z.B. in der Regelungstechnik sowie auch numerischen Verfahren, z.B. Newton-Verfahren (später im Kurs)

4. Nullstellen nichtlinearer Funktionen (Newton)

Bisektion: Binäre Suche

Bsp "High-Low Spiel": Gegeben eine Obergrenze n wird gesucht eine unbekannte natürliche Zahl z mit $0 < z < n$. Ein Spieler macht einen Rateversuch r_1 und erhält als Information:

– zu hoch (high), falls $z < r_1$,
– zu niedrig (low), falls $z > r_1$,
– korrekt, falls z gefunden wurde.

Danach erfolgen ggf. so lange weitere Rateversuche, bis die Zahl gefunden wurde.

Man kann die unbekannte Zahl in höchstens $\log_2(n)$ Schritten erraten.

Bisektion: Intervallhalbierung

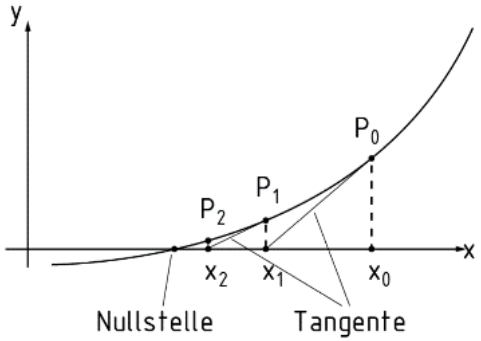
- Anwendung der Idee auf Nullstellensuche einer stetigen Funktion
- Beispiel: gesucht ist $\sqrt{3}$, d.h. Nullstelle von $f(x) := x^2 - 3$
- f ist stetig, d.h. aus $f(1) < 0$ und $f(2) > 0$ folgt, dass das Intervall $]1,2[$ eine Nullstelle von f enthalten muss.
- Die Zahlen im Intervall sind angeordnet, aber die Menge ist nicht endlich. Damit die Idee trotzdem funktioniert, bricht die Suche nicht nur dann ab, wenn das Resultat gefunden wurde, sondern auch, wenn das einschliessende Intervall klein genug ist.

Newton-Verfahren: Einführendes Beispiel

- Heronsches Näherungsverfahren: Startwert x_0, verbessertes $x_1, x_2, \ldots$.
- Wurzelberechnung $\sqrt{a}$ als Nullstellensuche von $f(x) = x^2 - a$:
  $x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)$
- Abbruch falls Residuum $r = a - x_k^2$ klein genug

Newton-Verfahren

- **Gegeben:** Intervall $I = [a, b]$ mit $f(a) \cdot f(b) < 0$
- **Idee:** finde Nullstelle von f aus der Linearisierung, d.h. aus Nullstellen der Tangenten im Punkt $x_0, x_1, \ldots$



- **Frage:** Welche Nullstelle hat die Linearisierung von f im Entwicklungspunkt x_k, gegeben durch $t_1(x) = f(x_k) + f'(x_k)(x - x_k)$?

Allgemeines Newtonverfahren für:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Es sei x^* die gesuchte Nullstelle, also $f(x^*)$ und x_k eine Iterierte des Newton-Verfahrens. Aus der einfachsten Taylorentwicklung t_0 von f am Entwicklungspunkt x^* gilt also mit dem entsprechenden Restglied und einer Zwischenstelle z:

$$f(x_k) = f(x^*) + \frac{f'(z)}{1!}(x_k - x^*)$$

Daraus folgt für das Residuum $|f(x_k)|$ also:

$$|f(x_k)| = |f'(z)| \cdot |x_k - x^*|$$

Aus einem kleinen Residuum kann man i.a. also nicht direkt auf einen kleinen Fehler $|x_k - x^*|$ schliessen.

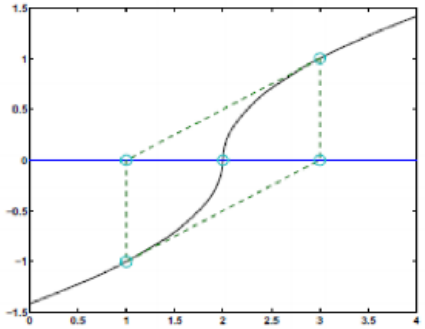
Besonderheiten des Newton-Verfahrens

Leider ist das Newtonverfahren nicht global konvergent (und im allgemeinen auch gar nicht überall definiert).

Beispiel:

Hinreichende Voraussetzung für

- lokale Konvergenz in $I = [a, b]$ mit $f(a) \cdot f(b) < 0$:
- strikte Konvexität (bei zweimal differenzierbaren Funktionen äquivalent zu $f'' > 0$)



Newton-Verfahren zur k-ten Wurzel einer Zahl a > 0

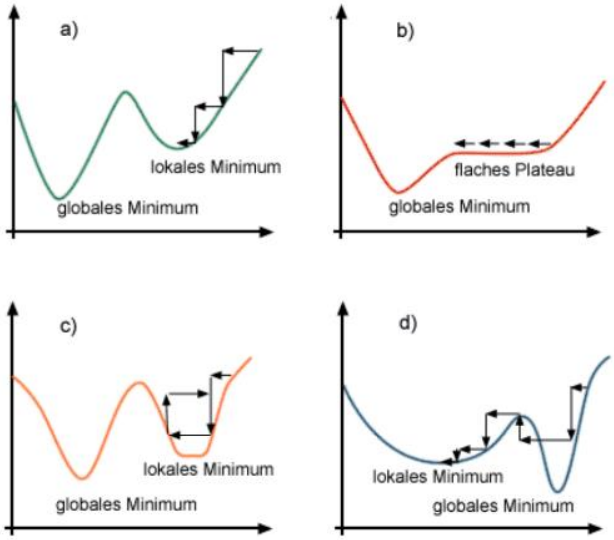
$$f(x) = x^k - a \qquad \Rightarrow$$

$$x_{n+1} = \frac{1}{k}\left((k-1)x_n + \frac{a}{x_n^{k-1}}\right)$$

5. Newton-Verfahren (Vertiefung)

Minimierung einer differenzierbaren Funktion: Hinweis

- Konvergenz zu einem lokalen anstatt eines globalen Minimums (a)
- Stagnation ohne Konvergenz bei einer Funktion mit weiten Plateaus möglich. (b)
- Nicht-Konvergenz und Oszillationen (schwingung), z.B. bei steilen Schluchten, möglich (c)
- Verlassen guter Minima hin zu unerwünschten Extremwerten (d)



- Randwerte müssen gesondert geprüft werden.
- Spezialisierte Algorithmen existieren, siehe z.B. die Dokumentation der Matlab Optimization Toolbox.

Partielle Ableitung

$$f(x, y) = 2x^2 + 4y, \qquad \frac{\partial f}{\partial x}(x, y) = 4x,$$

$$\frac{\partial d}{\partial y}(x, y,) = 4$$

D-Linearisierung (**Tangentialebene** als Verallgemeinerung der Tangente)

$$t_1(x, y) = f(x_0, y_0) + \frac{\partial f}{\partial x}(x_0, y_0)(x - x_0) + \frac{\partial f}{\partial y}(x_0, y_0)(y - y_0)$$

Beobachtung: Nullstellen der Linearisierung $t_1(x, y) = 0$ sind eine Gerade. Um Nullstellensuche und Newton-Verfahren aus dem 1D zu verallgemeinern, braucht man zwei Funktionen in zwei Variablen; neue 2D-Newton-Iterierte als Schnittpunkt der zwei Geraden.

Jacobi-Matrix

$$f(x_1,x_2)=\begin{pmatrix}f_1(x_1,x_2)\\ f_2(x_1,x_2)\end{pmatrix}\rightarrow \quad J(x_1,x_2)$$
$$=\begin{bmatrix}\frac{\partial f_1}{\partial x_1}(x_1,x_2) & \frac{\partial f_1}{\partial x_2}(x_1,x_2)\\ \frac{\partial f_2}{\partial x_1}(x_1,x_2) & \frac{\partial f_2}{\partial x_2}(x_1,x_2)\end{bmatrix}$$

Mehrdimensionales Newton-Verfahren

$$x_{k+1}=x_k-[J(x_k)]^{-1}\cdot f(x_k)$$

Die Inverse $[J(x_k)]^{-1}$ wird nicht explizit ausgerechnet (und das MV-Produkt $[J(x_k)]^{-1}\cdot f(x_k)$ auch nicht so durchgeführt). Stattdessen berechnet man den Newton-Schritt d_k aus:

$$[J(x_k)]\cdot d_k=f(x_k)$$

(Gauss-Alg. für Lineares Gleichungssystem) und setzt:

$$x_{k+1}=x_k-d_k$$

Euklidische Länge von Vektor f

```
norm(f)
```

Gauss-Algorithmus, numerisch besser als

```
d=inv(J)*f        d=J\f        A\b=x
```

6. Lineare Gleichungssysteme Numerischer Gauss-Algorithmus

Gauss-Algorithmus und A = L · U

Ein illustratives Beispiel:

Gleichwertige Aussagen: Die Spalten von A sind linear unabhängig. Die Matrix A ist invertierbar. Das lineare Gleichungssystem ist eindeutig lösbar. $\det(A)\neq 0$

$$Ax=b,\qquad A=\begin{bmatrix}10^{-20} & 1 & 0\\ 1 & 0 & 1\\ 1 & 1 & 0\end{bmatrix},$$
$$b=\begin{bmatrix}1\\ 1000\\ 1\end{bmatrix},\qquad x=\begin{bmatrix}0\\ 1\\ 1000\end{bmatrix}$$

!! Matlabs Gauss-Algorithmus \ rechnet korrekt. Allerdings ist dies nicht eine direkte Umsetzung des in der Linearen Algebra vorgestellten Gauss-Algorithmus.

Ohne Modifikation auf dem Computer erhält man statt der obigen eine Lösung: $\tilde{x}=\begin{bmatrix}0\\1\\0\end{bmatrix}$

Nutzen der Faktorisierung A = L · U

Faktorisierung A = L · U wobei U obere Dreiecksmatrix; L enthält die bei den Umformungen benutzten Multiplikatoren.

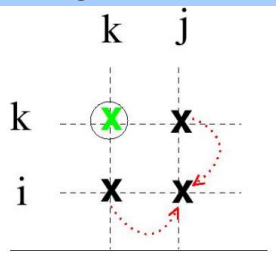
$$A=\begin{bmatrix}1 & -2 & -1\\ 2 & -1 & 1\\ 3 & -6 & 5\end{bmatrix},\qquad b=\begin{bmatrix}3\\0\\3\end{bmatrix},$$
$$L=\begin{bmatrix}1 & 0 & 0\\ 2/1 & 1 & 0\\ 3/1 & 0 & 1\end{bmatrix},$$
$$U=\begin{bmatrix}1 & -2 & -1\\ 0 & 3 & 3\\ 0 & 0 & -2\end{bmatrix}$$

Lösung von Ax = b, mit A = L · U:

1. Zwischenlösung y aus Ly = b, danach
2. x aus Ux = y.

Berechnung von L und U also altbekannt, per Gauss-Algorithmus, aber die neue Darstellung A = L · U erlaubt eine Fehleranalyse!

Berechnung von L und U mittels Gauss-Algorithmus



$$a_{ij}^{(k+1)}=a_{ij}^{(k)}-\frac{a_{ik}^{(k)}\cdot a_{kj}^{(k)}}{a_{kk}^{(k)}}$$

- Verbindung zur k-ten Spalte von L und Zeile von U:

$$l_{ik} := a_{ik}^{(k)}/a_{kk}^{(k)},\quad i=k+1,\dots,n,$$
$$u_{kj} := a_{kj}^{(k)},\quad j=k+1,\dots,n.$$

- $a_{kk}^{(k)}\neq 0$ ist das k-te 'Pivot'

Numerik eines linearen Gleichungssystems

Beurteilung einer Näherungslösung

1) Gegeben sei ein lineares Gleichungssystem $Ax=b$ mit theoretisch exakter Lösung $x=A^{-1}b$ sowie errechneter Näherungslösung $\tilde{x}$
2) Der Fehler $e:=x-\tilde{x}$ ist im Allgemeinen unbekannt.

Indirekte Fehlermessung per ***Residuum:*** $\boldsymbol{r=b-A\tilde{x}}$.

Beim Residuenvektor misst man die euklidische Länge (Matlab-Befehl `norm()`)

$$\|r\|\leq\|A\|\cdot\|x-\tilde{x}\|\rightarrow untere\ Schranke\colon \|x-\tilde{x}\|$$
$$=\|e\|\leq\frac{\|r\|}{\|A\|}$$

Relativer Fehler einer Näherungslösung

Abschätzung des relativen Fehlers (Fehlerschranke):

$$\frac{\|e\|}{\|x\|}\leq\underbrace{\|A\|\cdot\|A^{-1}\|}_{\kappa(A)}\cdot\frac{\|r\|}{\|b\|}$$

Eine Matrix A mit grosser Konditionszahl $\kappa(A)=\|A\|\cdot\|A^{-1}\|\gg 1$ nennt man schlecht konditioniert.

```
cond (A)
```

Gegenüber dem Newton-Verfahren vereinfacht sich die Diskussion, aufgrund des linearen Zusammenhangs von Fehler und Residuum sowie der einfachen Umkehrfunktion.

Analyse des illustrativen Anfangsbeispiels

- **Erkenntnis:** in der numerischen Umsetzung des Gauss-Algorithmus ist nicht nur das exakte Null-Pivot störend, sondern auch allgemein kleine Pivots.
- **Matlabs Gauss-Algorithmus \ numerisch korrekt, grundsätzlich immer diesen verwenden!** Naiver Gauss und einfache illustrative Programmbeispiele von heute allgemein nicht praktisch geeignet.

7. Ausgleichsrechnung (Regression)

Lineare Regression

Interpolation und Regression

- **Interpolation** konstruiert Näherungswerte einer Funktion zwischen Punkten, an denen die Funktion exakt bekannt ist (andere Lektion).
- **Regression** nimmt an, dass die bekannten Funktionswerte Fehler enthalten. Regression findet in einer gewählten Modellklasse (z.B. lineare Funktionen) dasjenige Modell, welches eine Fehlerfunktion (z.B. mittlere quadratische Abweichung) minimiert.

Abwägung bei Modellauswahl (sog. bias-variance tradeoff):

- ➢ **komplexeres Modell:** sensitiver gegenüber Datenänderungen, dafür besserer Fit der vorliegenden Daten.
- ➢ **Einfacheres Modell:** eventuell weniger guter Daten-Fit, dafür robust gegenüber Datenänderungen und in Voraussagen

Ausgleichsproblem

Einfache Beschreibung von komplizierteren Funktionen durch Polynome

```
%Generiere kompliziertere Beispielfunktion
x = [-2.5:0.1:2.5]'; y = tanh(x);
% Finde Koeffizienten von Fittingpolynom
grad=3; p = polyfit(x,y,grad);
% Evaluiere gefundenes Polynom
yy = polyval(p,x);
```

Bemerkenswert: der Polynomgrad ist ohne Schwierigkeit veränderbar! Lineare Regression beinhaltet also mehr als nur die bis jetzt aus der Statistik bekannte Ausgleichsgerade.

Wichtige Erkenntnis: *linear* bezieht sich nicht auf Gerade, sondern auf Linearität des Modells in den zu bestimmenden Koeffizienten.

Ausgleichspolynome aus Minimierung der Fehlerquadratsumme

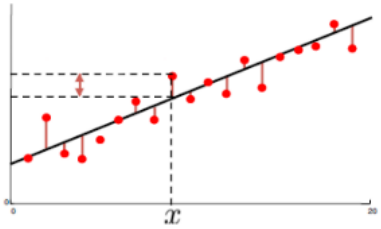
Modellformulierung

Gegeben: Punktpaare $(x_1, y_1), \ldots, (x_m, y_m)$ sowie vorgegebener Polynomgrad n, typisch: $m \gg n$.

Annahme: Messfehler in Ordinaten $y_1, \ldots, y_m$.

Gesucht: das beste Polynom p_n vom Höchstgrad n sodass Fehlerquadratsumme

$$F := \sum_{i=1}^{m} (p_n(x_1) - y_i)^2$$



minimiert wird. p minimiert also Länge des Residuenvektors $r = y - A \cdot p$.

Steigung m und Achsenabschnitt b der optimalen Geraden aus:

$$\min \left\| \underbrace{\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & 1 \\ x_n & 1 \end{bmatrix}}_{A = Vandermonde} \cdot \underbrace{\begin{pmatrix} m \\ b \end{pmatrix}}_{p} - \underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{y} \right\|^2$$

Designmatrix A

hat Schlüsselrolle in Koeffizientenberechnung für $y = ax^3 + bx^2 + cx + d$:

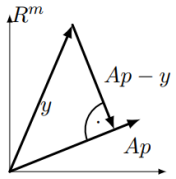
```
A=[x.^3,x.^2,x,ones(size(x))]
p=A'*A\A'*y
```

-> p=[a,b,c,d]

- ➢ **Bemerkung:** Die Matrix $(A^T \cdot A)$ ist quadratisch und symmetrisch.
- ➢ $(A^T \cdot A)$**invertierbar** falls **Spalten von A linear unabhängig**
- ➢ Falls $(A^T \cdot A)$ invertierbar: **Normalengleichung eindeutig lösbar**

Verbesserte Berechnung von p

Orthogonalität



- **Interpretation via Skalarprodukt:** das minimale Residuum ist senkrecht zu den Spalten von A; erklärt den Begriff Normalengleichung.
- **Orthogonalprojektion von y auf die Spalten der Designmatrix A** theoretisch äquivalent aber numerisch vorteilhafter.
- Matlabs \ implementiert diese Idee, automatische **Verwendung falls A nicht quadratisch**.

```
p=A\y
```

8. Polynom-Interpolation

Polynom-Interpolation als lineares Gleichungssystem

Vandermonde-Matrix

Polynom vom Grad n hat $(n+1)$ Koeffizienten.

```
% Interpolationsdaten
x=[0:.5:2]; y=[1 1 0 0 3]';
% Berechnung der Koeffizienten des Interpolationspolynoms
V=vander(x)
p=V\y; %Gauss
% Vergleichsplot
```

```
xx=-1:0.1:3; yy=polyval(p,xx);
plot(x,y,'b*',xx,yy,'r')
```

Man kann zeigen, dass für die Determinante einer $(n+1)\times(n+1)$ Vandermonde-Matrix V zu den Interpolationsknoten $x_0, x_1, \ldots, x_n$ gilt:

$$\boldsymbol{det(V) = (x_n - x_{n-1}) \cdot (x_n - x_{n-2}) \cdot \ldots \cdot (x_n - x_0) \cdot (x_{n-1} - x_{n-2}) \cdot \ldots \cdot (x_1 - x_0)}$$

das Produkt umfasst alle möglichen Differenzen als Faktoren genau einmal. Was folgt daraus für die eindeutige Lösbarkeit der Interpolationsaufgabe, falls $x_i \neq x_j$, $i \neq j$, erfüllt ist? $\det(V) \neq 0 \rightarrow$ Polynom eindeutig.

Konditionszahlen von Vandermonde-Matrizen ab 7-10 nicht mehr sehr gut.

Polynom-Interpolation nach Lagrange

Lagrange-Polynome INTERPOLATIONSPOLYNOM

$$p(x) = y_0 \cdot l_0(x) + y_1 \cdot l_1(x) + \cdots + y_n \cdot l_n(x)$$

$$l_i(x) = \frac{(x - x_0) \cdot \ldots \cdot (x - x_{i-1}) \cdot (x - x_{i+1}) \cdot \ldots \cdot (x - x_n)}{(x_i - x_0) \cdot \ldots \cdot (x_i - x_{i-1}) \cdot (x_i - x_{i+1}) \cdot \ldots \cdot (x_i - x_n)}$$

Verifizieren Sie, dass die Lagrangepolynome von $f(x) = 1/x$ mit den Stützstellen $x_0 = 1, x_1 = 2, x_2 = 3, x_3 = 4$ gegeben sind durch

$$\ell_0(x) = \frac{(x-2)(x-3)(x-4)}{(1-2)(1-3)(1-4)} = -\frac{1}{6}(x^3 - 9x^2 + 26x - 24)$$

$$\ell_1(x) = \frac{(x-1)(x-3)(x-4)}{(2-1)(2-3)(2-4)} = \frac{1}{2}(x^3 - 8x^2 + 19x - 12)$$

$$\ell_2(x) = \frac{(x-1)(x-2)(x-4)}{(3-1)(3-2)(3-4)} = -\frac{1}{2}(x^3 - 7x^2 + 14x - 8)$$

$$\ell_3(x) = \frac{(x-1)(x-2)(x-3)}{(4-1)(4-2)(4-3)} = \frac{1}{6}(x^3 - 6x^2 + 11x - 6)$$

$$\begin{aligned} p(x) &= 1 \cdot \ell_0(x) + \frac{1}{2}\ell_1(x) + \frac{1}{3}\ell_2(x) + \frac{1}{4}\ell_3(x) \\ &= -\frac{1}{24}x^3 + \frac{5}{12}x^2 - \frac{35}{24}x + \frac{25}{12}. \end{aligned}$$

Spline-Interpolation

Stückweise lineare Interpolation

Beispiel einer stückweise linearen Interpolation mit `interp1`

```
%Interpolationsknoten
x = 1:8;
%Interpolationswerte
y = zeros(length(x),1); y(3)= 0.3;
%Punkte zum Plotten
xx = 0:.25:10;
%Stückweise lineare Interpolation an den
xx yy = interp1(x,y,xx,'linear');
```

Stückweise kubische Splines

- Nachteil der stückweisen linearen Interpolation: Funktion hat 'Spitzen'.
- **Idee:** Verwendung einer kubischen Funktion auf jedem Teilintervall.
- n Teilintervalle, also $4n$ Koeffizienten (davon $2n$ bestimmt durch Interpolationsbedingungen wie im linearen Fall)
- Zwei Freiheitsgrade pro Teilabschnitt mehr als stückweise lineare Interpolation; benutzt um Funktion 'runder' zu machen. Stetige erste und zweite Ableitung, an jedem von $n-1$ Übergängen
- Wahl von 2 zusätzlichen Bedingungen für *eindeutigen* kubischen Spline, verschieden Varianten möglich, siehe **Matlabs `spline` und `pchip`**

9. Numerische Integration (Quadratur)

Numerische Quadratur nach Newton-Cotes: Quadraturformeln

Typische numerische Quadraturformel zur Approximation von:

$$\int_a^b f(x)dx \approx I_n := \sum_{j=0}^{n} f(x_i) \cdot w_j$$

Mit $n+1$ Integrationsknoten (Quadraturknoten)

$$a \leq x_0 < x_1 < \cdots x_n \leq b$$

Und zugehörigen Gewichten w_j (nicht immer positiv, ab $n \geq 8$).

Quadraturformeln nach Newton-Cotes: äquidistante Knotenwahl mit:

$$h := x_1 - x_0 = x_2 - x_1 = \cdots = x_n - x_{n-1}$$

Mittelpunktregel, auch Rechtecksregel ($n = 0$, Interpolation mit Konstante):

$$M_0 := (b-a) \cdot f\left(\frac{a+b}{2}\right)$$

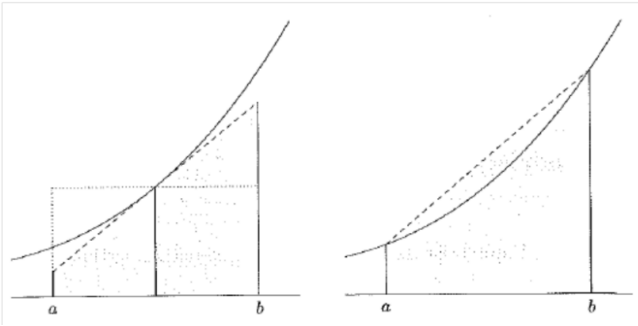
Trapezregel (n=1, Interpolation mit Gerade):

$$T_1 := \frac{b-a}{2} \cdot f(a) + \frac{b-a}{2} \cdot f(b)$$

Simpsonregel ($n = 2$, Interpolation mit Parabel):

$$S_2 := \frac{b-a}{6} \cdot f(a) + 4 \cdot \frac{b-a}{6} \cdot f\left(\frac{a+b}{2}\right) + \frac{b-a}{6} \cdot f(b)$$

- Illustration: Mittelpunktregel (links) und Trapezregel (rechts)



Konstruktion von Quadraturformeln

Quadraturformeln aus der Interpolation

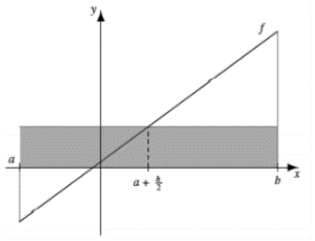
Idee: anstatt Originalfunktion f, integriere Interpolationspolynom (welches f gut approximiert)

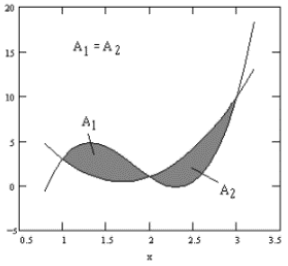
$$\int_a^b f(x)dx \approx \int_a^b p(x)dx = \sum_{j=0}^{n} y_j \underbrace{\int_a^b \ell_j(x)dx}_{=:w_j}$$

Mit $\boldsymbol{n+1}$ **Knoten** werden **alle Polynome vom Grad** $\boldsymbol{n}$ **exakt integriert!**

- Verifizieren Sie analytisch/geometrisch, dass die Mittelpunktregel nicht nur konstante sondern auch lineare Polynome exakt integriert
- Allgemein (folgt durch eine Symmetriebetrachtung auf dem Standardintervall $[-1,1]$): Eine Newton-Cotes-Formel mit $n+1$ Knoten, wobei $n+1$ ungerade, integriert alle Polynome vom Grad $n+1$ exakt.

Bsp: die Simpsonregel (5) berechnet nicht nur bestimmte Integrale quadratischer sondern auch kubischer Polynome exakt





Alternative Konstruktion der Gewichte

Alternativ kann man die Gewichte auch aus einem linearen Gleichungssystem ausrechnen, $w = A\backslash b$.
Man benutzt, dass alle Interpolationspolynome vom Höchstgrad n mit $n+1$ Interpolationsknoten exakt integriert werden.

- Beispiel Trapezregel, diese hat die Form

$$T_1 := w_0 \cdot f(a) + w_1 \cdot f(b).$$

Wenn die Polynome 1 und x exakt integriert werden sollen, so muss gelten

$$\begin{bmatrix} 1 & 1 \\ a & b \end{bmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} = \begin{pmatrix} \int_a^b 1dx \\ \int_a^b xdx \end{pmatrix} = \begin{pmatrix} b-a \\ b^2/2 - a^2/2 \end{pmatrix}$$

Man findet dann durch Lösen

$$T_1 := \frac{b-a}{2} \cdot f(a) + \frac{b-a}{2} \cdot f(b).$$

- Die Simpsonregel hat die Form

$$S_2 := w_0 \cdot f(a) + w_1 \cdot f\left(\frac{a+b}{2}\right) + w_2 \cdot f(b).$$

Wenn die Polynome 1, x und x^2 exakt integriert werden sollen, so muss gelten

$$\begin{bmatrix} 1 & 1 & 1 \\ a & \frac{a+b}{2} & b \\ a^2 & (\frac{a+b}{2})^2 & b^2 \end{bmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} \int_a^b 1dx \\ \int_a^b xdx \\ \int_a^b x^2dx \end{pmatrix} = \begin{pmatrix} b-a \\ b^2/2 - a^2/2 \\ b^3/3 - a^3/3 \end{pmatrix}$$

und daraus

$$S_2 := \frac{b-a}{6} \cdot f(a) + 4 \cdot \frac{b-a}{6} \cdot f\left(\frac{a+b}{2}\right) + \frac{b-a}{6} \cdot f(b).$$

Beispiel:

- *Trapezregel* ($n = 1$, Interpolation mit Gerade):

$$T_1 := \frac{b-a}{2} \cdot f(a) + \frac{b-a}{2} \cdot f(b)$$

Berechnung der Gewichte:

$$\begin{aligned} \int_a^b \ell_0(x)dx &= \int_a^b \frac{x-b}{a-b}dx \\ &= \frac{1}{a-b}\left[\frac{1}{2}x^2 - bx\right]_a^b = \frac{1}{a-b}\left[\frac{b^2-a^2}{2} - (b^2-ab)\right] = \frac{b-a}{2} \\ \int_a^b \ell_1(x)dx &= \int_a^b \frac{x-a}{b-a}dx \\ &= \frac{1}{b-a}\left[\frac{1}{2}x^2 - ax\right]_a^b = \frac{1}{b-a}\left[\frac{b^2-a^2}{2} - (ab-a^2)\right] = \frac{b-a}{2} \end{aligned}$$

Simpsonregel ($n = 2$):

$$S_2 := \frac{b-a}{6} \cdot f(a) + 4 \cdot \frac{b-a}{6} \cdot f\left(\frac{a+b}{2}\right) + \frac{b-a}{6} \cdot f(b)$$

Berechnung der Gewichte:

$$\begin{aligned} \int_a^b \ell_0(x)dx &= \int_a^b \frac{(x-(a+b)/2))(x-b)}{(a-(a+b)/2))(a-b)}dx \\ &= \frac{1}{(a-b)^2}\left[b^2x - \frac{3bx^2}{2} + abx + \frac{2x^3}{3} - \frac{ax^2}{2}\right]_a^b = \frac{b-a}{6} \\ \int_a^b \ell_1(x)dx &= \int_a^b \frac{(x-a)\ (x-b)}{((a+b)/2-a)((a+b)/2-b))}dx \\ &= \frac{2}{3}(b-a) \\ \int_a^b \ell_2(x)dx &= \int_a^b \frac{(x-a)(x-(a+b)/2)}{(b-a)(b-(a+b)/2)}dx \\ &= \frac{b-a}{6} \end{aligned}$$

AlternativkannmandieGewichteauchauseinemlinearen Gleichungssystemausrechnen,**w=A\b**

Die Drei-Achtel Regel integriert Polynome vom Grad ≤ 3 auf dem Intervall $[a,b]$:

$$Q_3 := \frac{b-a}{8} \cdot f(a) + 3 \cdot \frac{b-a}{8} \cdot f\left(\frac{2a+b}{3}\right) + 3 \cdot \frac{b-a}{8} \cdot f\left(\frac{a+2b}{3}\right) + \frac{b-a}{8} \cdot f(b)$$

Beispiel:

Man benutzt, dass alle **Interpolationspolynome vom Höchstgrad n mit n + 1 Interpolationsknoten** exakt integriert werden. Die Gewichte w_j mit j = 0,2, ..., n lassen sich so bestimmen, dass die Quadraturformel für n + 1 linear unabhänginge Funktionen und ihre Linearkombinationen die **exakten Integralwerte liefert.**
Wir **berechnen diese n + 1 Integral** und erhalten:

$$\begin{aligned} f(x) = 1: &\quad (1 \cdot w_0 + 1 \cdot w_1 + \ldots 1 \cdot w_n) = \int_a^b 1dx = b-a \\ f(x) = x: &\quad (x_0 \cdot w_0 + x_1 \cdot w_1 + \ldots x_n \cdot w_N) = \int_a^b xdx = \frac{b^2-a^2}{2} \\ f(x) = x^2: &\quad (x_0^2 \cdot w_0 + x_1^2 \cdot w_1 + \ldots x_n^2 \cdot w_N) = \int_a^b x^2dx = \frac{b^3-a^3}{3} \\ &\vdots \end{aligned}$$

$f(x) = x^n$: $(x_0^n \cdot w_0 + x_1^n \cdot w_1 + ...x_n^n \cdot w_N) = \int_a^b x^n dx = \frac{b^{n+1} - a^{n+1}}{1+n}$

In Matrixform geschriebe → das Gleichungssystem:

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_0 & x_1 & x_2 & \cdots & x_n \\ x_0^2 & x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & & & & \\ x_0^n & x_1^n & x_2^n & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} b-a \\ \frac{1}{2}(b^2-a^2) \\ \frac{1}{3}(b^3-a^3) \\ \vdots \\ \frac{b^{n+1}-a^{n+1}}{1+n} \end{pmatrix}$$

Trapezregel, diese hat die Form

$$T_1 := w_0 \cdot f(a) + w_1 \cdot f(b).$$

Wenn die Polynome 1 und x exakt integriert werden sollen und $x_0 = a$ und $x_1 = b$, so muss gelten

$$\begin{bmatrix} 1 & 1 \\ a & b \end{bmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} = \begin{pmatrix} \int_a^b 1dx \\ \int_a^b x dx \end{pmatrix} = \begin{pmatrix} b-a \\ \frac{b^2}{2} - \frac{a^2}{2} \end{pmatrix}$$

Man findet dann durch Lösen

$$T_1 := \frac{b-a}{2} \cdot f(a) + \frac{b-a}{2} \cdot f(b).$$

Die *Simpsonregel* hat die Form

$$S_2 := w_0 \cdot f(a) + w_1 \cdot f\left(\frac{a+b}{2}\right) + w_2 \cdot f(b).$$

Wenn die Polynome 1, x und x^2 exakt integriert werden sollen, so muss gelten

$$\begin{bmatrix} 1 & 1 & 1 \\ a & \frac{a+b}{2} & b \\ a^2 & (\frac{a+b}{2})^2 & b^2 \end{bmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} \int_a^b 1dx \\ \int_a^b x dx \\ \int_a^b x^2 dx \end{pmatrix} = \begin{pmatrix} b-a \\ \frac{b^2}{2} - \frac{a^2}{2} \\ \frac{b^3}{3} - \frac{a^3}{3} \end{pmatrix}$$

und daraus

$$S_2 := \frac{b-a}{6} \cdot f(a) + 4 \cdot \frac{b-a}{6} \cdot f\left(\frac{a+b}{2}\right) + \frac{b-a}{6} \cdot f(b).$$

Nehmen wir zur Illustration das Intervall [0,1] und $n+1 = 3$, dann erhalten wir $x_0 = 0, x_1 = 0.5$ und $x_2 = 1$. Eingesetzt in das Gleichungssystem erhalten wir:

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0.5 & 1 \\ 0 & 0.25 & 1 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \end{pmatrix}$$

.

Wir erhalten $w_0 = \frac{1}{6}, w_1 = \frac{2}{3}$ und $w_2 = \frac{1}{6}$. Für ein beliebiges Intervall [a,b] lautet die Quadraturformel wie in (8).

Zusammengesetzte Newton-Cotes Formeln

Stückweise Integration

Zusammengesetzte Integrationsformeln: Unterteilung des Intervalls $[a, b]$ in n Teilintervalle der Grösse $h = \frac{b-a}{n}$, Anwendung der Originalformeln auf jedes Teilintervall:

- *Mittelpunktregel* ⇒ *Zusammengesetzte Mittelpunktregel*:

$$M_0 = (b-a) \cdot f\left(\frac{a+b}{2}\right) \Rightarrow$$

$$M_0(h) = h \cdot \left[f(\frac{a+(a+h)}{2}) + f(\frac{(a+h)+(a+2h)}{2}) + \ldots + f(\frac{(a+(n-1)h)+b}{2})\right]$$

- *Trapezregel* ⇒ *Zusammengesetzte Trapezregel*:

$$T_1 = \frac{b-a}{2} \cdot f(a) + \frac{b-a}{2} \cdot f(b) \Rightarrow$$

$$T_1(h) := \frac{h}{2} \cdot [f(a) + 2f(a+h) + \ldots + 2f(a+(n-1)h) + f(b)]$$

- *Simpsonregel* ⇒ *Zusammengesetzte Simpsonregel*:

$$S_2 = \frac{b-a}{6} \cdot f(a) + 4 \cdot \frac{b-a}{6} \cdot f\left(\frac{a+b}{2}\right) + \frac{b-a}{6} \cdot f(b) \Rightarrow$$

$$S_2(h) := \frac{h}{6} \cdot \left[f(a) + 4f(\frac{a+(a+h)}{2}) + 2f(a+h) + \ldots + f(b)\right]$$

10. Gewöhnliche Differentialgleichungen

Analytische Lösung

Falls eine lineare Differentialgleichung eine analytische Lösung hat, kann man sie in Matlab mit der Funktion dsolve explizit lösen.

DGL der 1. Ordnung

Will man $y' = 2xy$ lösen, so schreibt man:

```
y=dsolve('Dy=2*x*y','x')
y =
C2*exp(x^2)
```

- Die Gleichung muss als String übergegeben werden. Dabei wird das Differential als Dy bezeichnet.
  Die unabhängige Variable x, muss man ebenfalls als String angeben.
- Die Ausgabe ist vom Typ *symbolic*. Schauen Sie sich die Matlab-Hilfe an, wenn Sie mehr zu diesem Datentyp wissen wollen.
- $C*$ ist immer eine Integrationskonstante.

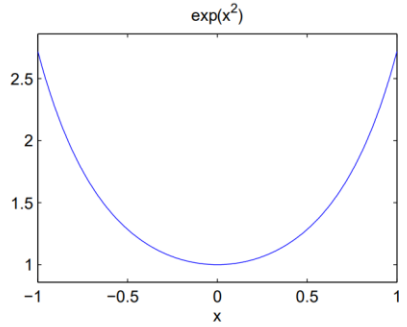
DGL der 1. Ordnung mit Anfangsbedingung

Hat man eine Anfangsbedingung, z.B. $y(0) = 1$ vorgegeben, so erhält man die explizite Lösung ohne Integrationskonstanten:

```
y=dsolve('Dy=2*x*y','y(0)=1','x')
y =
exp(x^2)
```

Diese Lösung kann man auch graphisch mit der Funktion ezplot darstellen. Dabei muss man Intervallgrenzen für x angeben.

```
ezplot(y, [-1,1])
```



Möchte man den numerischen Wert von z.B $Ay(0.5)$ wissen, tippt man

```
double( subs(y,'x',0.5) )
```

DGL höherer Ordnung

$$y'' + y' = 2y, \quad y(0) = 1,\ y'(0) = 1$$

```
y=dsolve('D2y + Dy =2*y', 'y(0)=1', 'Dy(0)=1', 'x')
```

$$y''' = x, \quad y(0) = 1,\ y(1) = 1,\ y(-1) = 1$$

```
y=dsolve('D3y =x','y(0)=0','y(1)=1','y(-1)=1','x')
```

System von DGL

$$y_1' = y_2; \quad y_2' = y_1, \quad \text{mit} \quad y_1(0) = 100;\ y_2(0) = 50$$

```
y=dsolve('Dy1 =y2','Dy2=-y1','y1(0)=100', 'y2(0)=50','x')
```

Die Ausgabe ist von Typ *structure*. Die beiden Ergebnisse erhält man folgendermaßen:

```
y.y1
y.y2
```

Man kann die beiden Funktionen auch zusammen darstellen. Leider kann man bei ezplot nicht auf der üblichen Weise die Farbe ändern.

```
ezplot(y.y1, [0 20])
hold on
h=ezplot(y.y2, [0,20]);
set(h, 'color', [1 0 0])
hold off
```

Beispiel

1. Lösen Sie die Differentialgleichung

$$y' = \frac{1}{1-x}y + x - 1 \quad \text{mit} \quad y(2) = 0$$

einmal mit und ohne Anfangswertbedingung. Sie können auch eine andere Anfangswertbedingung einsetzen.
Plotten Sie die Funktion mit a) ezplot und b) plot

1.3 Numerische Lösung

Matlab stellt zur numerischen Lösung bei Systemen gewöhnlicher Differentialgleichungen verschiedene Funktionen mit verschiedenen Eigenschaften zur Verfügung:

- ode45, Dormand-Rince Verfahren (Runge-Kutta)
- ode23, Bogacki-Shampine Verfahren (Runge-Kutta)
- ode113, Adams-verfahren
- ode15s, BDF Verfahren
- ode23s, Rosenbrock Verfahren
- ode23t, Trapez-Regel
- ode23tb, TR-BDF-Verfahren

Beispiel Funktion

[x,y] = ode45(@Fktname, Intervall, Anfangswerte, Optionen, FktInput);

1. **Funktionsname** Der Name einer m.Funktion, die im einfachsten Fall nur eine Gleichung beschreibt. Der Funktionskopf hat im einfachsten Fall die Form:

   function dy = Funktionsname(x,y)

   Die Argumente x und y müssen auch dann angegeben werden, wenn sie zur Berechnung der Ableitungen nicht benötigt werden. Diese beiden Argumente sind Spaltenvektoren. Der Funktionswert, hier dy genannt, muss die Ableitungen der Differentialgleichungen liefern. **dy muss ein Spaltenvektor sein**.

2. **x-Intervall**, ein Zeilen- oder Spalten-Vektor der Länge 2, der das Integrationsintervall festlegt.

3. **Anfangswerte** als Zeilen- oder Spalten-Vektor, der die Anfangswerte des Systems festlegt.

4. **Optionen** optional, Struktur, die die Eigenschaften von ode45 ändern. Siehe Hilfe ode45 und odeset. Wir werden die Eigenschaften nicht ändern.

5. **FktInput**, optional. Falls die m-Funktion noch zusätzliche Inputargumente benötigt, kann man diese hier angeben, siehe Beispiel 3.

Funktion 1

Wir lösen jetzt noch mal erneut die DGL:

$$y' = 2xy, \quad \text{mit} \quad y(-1) = \exp(1)$$

Die Funktion muss man unter DGL1.m abspeichern.

```
function dy=DGL1(x,y)
dy=2*x*y;
```

Die Gleichung soll im Intervall [-1 1] gelöst werden. Den Anfangswert, exp(1), ist der Wert, der am Anfang gelten soll. Also $y(-1) = exp(1)$. Das bedeutet, man kann nicht die Bedingung y(0)=1 angeben.

```
[x1,y1]=ode45(@DGL1,[-1 1],exp(1));
figure
plot(x1,y1)
```

Funktion 2 Räuber-Beute

piel 2 - Räuber-Beute Ein klassisches Modell zur Beschreibung von zwei einander bedingenden Populationsgrößen sind die Lotka-Volterra Gleichungen. Es handelt sich um ein System aus zwei gekoppelten Differentialgleichungen:

$$\frac{dP}{dt} = P(t)(r_P + c_P R(t))$$
$$\frac{dR}{dt} = R(t)(r_R + c_R P(t))$$

mit

$P(t)$	Populationsgröße der Beute zum Zeitpunkt t
r_P	konstante Reproduktionsrate der Beute, wenn keine Räuber anwesend sind (> 0, da die ungestörte Population wächst)
c_P	Änderung der Beute-Population pro Räuber (< 0, da die Population durch jeden Räuber verkleinert wird)
$R(t)$	Populationsgröße der Räuber
r_R	konstante Reproduktionsrate der Räuber, wenn keine Beute anwesend ist (< 0, da die Räuber ohne Beute sterben)
c_R	Änderung der Räuber-Population pro Beutetier (> 0, da durch jedes Beutetier Geburten ermöglicht werden)

Wir simulieren jetzt das Populationswachstum mit diesen Differentialgleichungen. Dabei setzen wir $r_P = 0.08; c_P = -0.002; r_R = -0.2; c_R = 0.0004$; und schreiben die m-Funktion:

```
function dbr=raeuberbeute(t,br)
dbr=zeros(2,1);
rP=0.08;
cP=-0.002;
rR=-0.2;
cR=0.0004;
dbr(1)=rP*br(1)+cP*br(1)*br(2);
dbr(2)=rR*br(2)+cR*br(1)*br(2);
```

Wir schauen ins einen Zeitraum von 100 Tagen an. Die Anfangspopulation der Beute ist 500 und die der Räuber 20.

```
[x,rb]=ode45(@raeuberbeute,[0,100],[500 20]);
plot(x,rb(:,1))
hold on
plot(x,rb(:,2),'r')
legend('Beute','Raeuber')
xlabel('Zeit in Tage')
ylabel('Populationsgroesse')
hold off
```

Man kann auch die Populationsgrößen gegeneinander auftragen.

```
figure plot(rb(:,1),rb(:,2))
xlabel('Populationsgroesse der Beute')
ylabel('Populationsgroesse der Raeuber')
```

Funktion 3 erweiterte Eingabeargumente

Es ist nartürlich lästig immer wieder eine ganze Datei zu ändern, falls man Werte in seiner m-Funktion variieren möchte. Wir schreiben eine erweiterte Funktion für das Räuber-Beute Beispiel.

```
function dbr=raeuberbeute2(t,br,werte)
dbr=zeros(2,1);
rP=werte(1);
cP=werte(2);
rR=werte(3);
cR=werte(4);
dbr(1)=rP*br(1)+cP*br(1)*br(2);
dbr(2)=rR*br(2)+cR*br(1)*br(2);
```

Damit können wir jetzt ganz einfach die Werte variieren.

```
rP=0.08;
cP=-0.002;
rR=-0.2;
cR=0.0004;
[x,rb]=ode45(@raeuberbeute2,[0,100],[500 20],[],[rP cP rR cR]);
```

Variieren Sie die Eingabeparameter!